PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2000-032388
(43)Date of publication of application : 28.01.2000

(51)Int.Cl. H04N 5/91
G11B 20/12
H04N 5/92

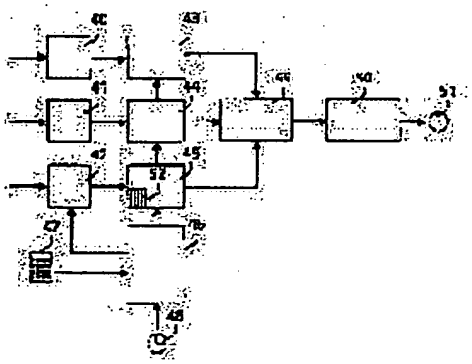
(21)Application number : 11-140504
(22)Date of filing : 20.05.1999

(71)Applicant : DEUTSCHE THOMSON BRANDT GMBH
(72)Inventor : SCHILLER HARALD
WINTER MARCO

(30)Priority
Priority number : 98 19822975 Priority date : 25.05.1998 Priority country : DE

(54) METHOD AND DEVICE FOR RECORDING AND REPRODUCING OF VIDEO AND/OR AUDIO SIGNAL

(57)Abstract:
PROBLEM TO BE SOLVED: To provide a device which separately records additional information data without changing a recording format by converting data for an additional information item into a subpicture unit and storing them in a subpicture data pack.
SOLUTION: Data for an additional information item is converted into a subpicture unit and is stored in a subpicture data pack. In this device, a user can input a desired title, for example, to be recorded at the same time by a key board unit 47. In a microcontroller 46, inputted data are logically ordered and transferred to a buffer storage device 42. The data stored in the buffer storage device 42 are corrected by a subpicture coding unit 45. For the inputted data, the subpicture unit is generated by the subpicture coding unit 45. The subpicture unit is transferred next to a data formatting unit 49.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号
特開2000－32388
（P2000－32388A）
(43)公開日　平成12年１月28日(2000.1.28)

(51)Int.Cl.'	識別記号	FI	テーマコード'（参考）
H04N 5/91		H04N 5/91	E
G11B 20/12		G11B 20/12	
	103		103
H04N 5/92		H04N 5/92	H

審査請求　未請求　請求項の数11　ＯＬ　（全 25 頁）

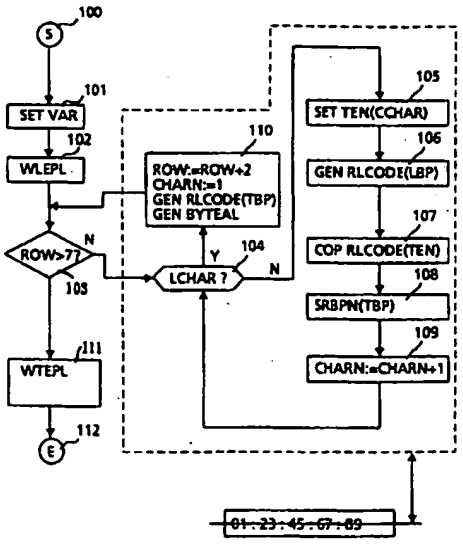
(21)出願番号　特願平11－140504
(22)出願日　平成11年５月20日(1999.5.20)
(31)優先権主張番号　１９８２２９７５：５
(32)優先日　平成10年５月25日(1998.5.25)
(33)優先権主張国　ドイツ（ＤＥ）

(71)出願人　595033034
ドイチェ　トムソン－ブラント　ゲーエムベーハー
Ｄｅｕｔｓｃｈｅ　Ｔｈｏｍｓｏｎ－Ｂｒａｎｄｔ　ＧｍｂＨ
ドイツ連邦共和国　デー－78048　ヴィリンゲン－シュヴェニンゲン　ヘルマン－シュヴェーアー－シュトラーセ　３
(72)発明者　ハーラルト　シラー
ドイツ連邦共和国，30539　ハノーヴァー，アップフェルガルテン　11
(74)代理人　100070150
弁理士　伊東　忠彦　（外１名）

最終頁に続く

(54)【発明の名称】　ビデオ及び／又はオーディオ信号の記録及び再生用の方法並びに装置

(57)【要約】
【課題】　本発明は、記録フォーマットを変更することなく、付加情報データを別個に記録するビデオ／オーディオ信号の記録・再生方法並びに装置の提供を目的とする。
【解決手段】　本発明によれば、再生中にビデオピクチャーに連続的に挿入される付加情報項目、例えば、タイトル情報項目が記録される。付加情報項目は、実行可能なサブピクチャーユニットに変換され、ビデオ／オーディオ信号用のデータパックの他に対応したサブピクチャーデータパックに記録される。



【特許請求の範囲】

【請求項１】　付加情報項目、特に、ビデオ及び／又はオーディオ信号の再生中にビデオピクチャーへ連続的に挿入するため役立つタイトル情報項目がビデオ及び／又はオーディオ信号に加えて記録されるビデオ及び／又はオーディオ信号の記録及び再生方法において、
上記付加情報項目に対するデータがサブピクチャーユニットに変換され、
上記サブピクチャーユニットは、記憶媒体上の上記ビデオ及び／又はオーディオ信号に対するデータパックの他に、少なくとも１個のサブピクチャーデータパックに記録されることを特徴とする方法。

【請求項２】　上記サブピクチャーユニットは、上記付加情報項目を表現する挿入用の圧縮ビットマップと、挿入用の制御コマンドを含むテーブルとを有し、
再生モードにおいて、上記記録されたサブピクチャーユニットは、上記圧縮ビットマップを復号化し、上記付加情報項目が上記ビデオピクチャーに挿入されるように上記制御コマンドを処理することにより変換される、請求項１記載の方法。

【請求項３】　上記ビットマップを圧縮するためランレングス符号化が使用され留請求項１又は２記載の方法。

【請求項４】　上記付加情報項目はオペレータによって入力されるか、若しくは、自動的に生成される、請求項１乃至３のうちいずれか一項記載の方法。

【請求項５】　上記付加情報項目は、テキスト文字、特に、ＡＳＣＩＩ文字の形式で入力されるか、若しくは、自動的に生成される、請求項１乃至４のうちいずれか一項記載の方法。

【請求項６】　上記ランレングス符号化されたビットマップを簡単に生成するため、ランレングス符号化されたテキスト／グラフィックス文字を備えたテーブルが設けられ、
上記付加情報項目に必要な上記テキスト／グラフィックス文字は、上記テーブルから選択され、対応したビットマップを形成するため合成される、請求項５記載の方法。

【請求項７】　テキスト／グラフィックス文字の画素行毎に上記テーブル内でランレングス復号化を行うため、
（ａ）上記画素行のランレングス符号を記述するためのデータワード、特に、ニブル又はビットの個数と、
（ｂ）上記画素行内の最初の文字画素以前の背景画素の個数と、
（ｃ）上記画素行内の最後の文字画素以降の背景画素の個数と、
（ｄ）先行する背景画素及び後続する背景画素が伴わない上記画素行のランレングス符号とが指定される、請求項４又は５記載の方法。

【請求項８】　複数の付加情報項目が同時に記録され、再生モード中に、上記複数の情報項目が交互に選択され、ビデオピクチャーに挿入される、請求項１乃至７のうちいずれか一項記載の方法。

【請求項９】　日付、時刻、日付及び時刻、再生時間、動作モード、タイトル、ユーザ定義特殊テキスト挿入項目、並びに、ユーザ定義特殊グラフィックス挿入項目の中の少なくとも一つの付加情報項目が記録される、請求項８記載の方法。

【請求項１０】　多数の付加情報項目が連続的に挿入されるべき順序が記録セクションに対しプログラムされる、請求項１乃至９のうちいずれか一項記載の方法。

【請求項１１】　付加情報項目、特に、ビデオ及び／又はオーディオ信号の再生中にビデオピクチャーへ連続的に挿入するため役立つタイトル情報項目がビデオ及び／又はオーディオ信号に加えて記録されるビデオ及び／又はオーディオ信号の記録及び再生装置において、
上記付加情報項目に対するデータから、上記ビデオ及び／又はオーディオ信号のためのデータパックの他に記憶媒体に記録するため役立つサブピクチャーユニットのための１個以上のデータパックを生成する装置と、
上記付加情報項目を表す挿入用の圧縮ビットマップと、上記挿入用の表示制御コマンドを含むテーブルとを収容するように上記サブピクチャーユニットを構成するエンコーディングユニットと、
再生中に、上記付加情報項目を上記ビデオピクチャーに挿入することによって可視化されるように、上記サブピクチャーユニットのための上記データパックに収容された上記サブピクチャーユニットを変換する上記記録されたサブピクチャーユニットのためのデコーディングユニットとを具備することを特徴とする記録及び再生装置。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、ビデオ及び／又はオーディオ信号を記録及び再生する方法に係わり、付加情報項目、特に、ビデオ信号及び／又はオーディオ信号の再生中にビデオピクチャーへ連続的に挿入するため役立つタイトル情報項目がビデオ及び／又はオーディオ信号に加えて記録される。

【０００２】本発明は、また、上記のビデオ及び／又はオーディオ信号を記録及び再生する方法が実施されるビデオ及び／又はオーディオ信号を記録及び再生する装置に関する。

【０００３】

【従来の技術】本発明は、付加情報項目、特に、再生中にビデオピクチャーへ連続的に挿入するため役立つがタイトル情報項目がビデオ及び／又はオーディオ信号に加えて記録されるビデオ及び／又はオーディオ信号の記録及び再生方法に基づく。磁気式記録及び再生用の方法並びに装置は、欧州特許明細書第ＥＰ　０　５５８　３２８号に記載されている。引用文献に記載された装置は、８ｍｍビデオレコーダに関する。この装置は、ビデオ信

号及びPCMオーディオ信号に加えてタイトルデータをビデオテープの傾斜トラックに記録する能力がある。タイトルデータは、PCMオーディオデータと同様にデジタル的に記録される。タイトルデータの他に、タイトルデータが提示される方法を決定する制御コードを記録することが可能である。タイトルデータを個別に記憶することにより、ユーザにタイトル挿入を行うべきか否かを決めるための選択の余地を与えることができる。

【0004】近年、市販され始めたDVDプレーヤの開発によって、例えば、テレビジョンセットの画面上にサブタイトルを表示するためのいわゆるサブ・ピクチャーデコーディング装置を設けることが知られている。サブピクチャーデコーディング装置の記述は、DVD規格、読み出し専用ディスク用DVD仕様、第3分冊、ビデオ仕様書、バージョン1．0，1996年8月発行の第5．4．3節サブピクチャーユニット（SPU）に記載されている。

【0005】かかるサブピクチャーデコーディング装置の詳細な説明は、欧州特許出願第EP－A－0　725541号に記載され、この引用文献には、DVDディスクの生産に関する説明中でプロフェッショナル分野のためのサブピクチャーユニット（SPU）に関する情報が含まれる。家電製品に属する殆どの装置において、一般的に、装置の状態情報項目及び動作指令等が出力ビデオピクチャーに挿入される。また、動作的制御のためのいわゆるオン・スクリーン・ディスプレイ（OSD）メニューの使用は普及している。一般的に、内蔵又は外付けROM及びRAMメモリを備えた専用マイクロコントローラ回路が上記のOSDメニュー又は他の状態表示を発生させるため使用される。これらの回路は、主として、表示可能な各文字毎にドットマトリックス状パターンがROMメモリに格納されるように設計される。表示可能な文字は、グラフィックス文字を含む必要があり、このグラフィックス文字を用いて、例えば、適切なバーチャートが合成され得る。OSDメニューの場合に、必要な文字コードが文字発生器に転送され、文字発生器は、ROMから関連したドットマトリックスデータを取得し、関連したRGB信号をピクチャー内の正しい場所に発生させ、ピクチャーに挿入する。この解決法は、（デジタルフレーム記憶装置を含む）デジタル信号処理を具備した装置に対し、画素データが文字発生器によって発生され、フレーム記憶装置の対応した場所にそのまま書き込まれるように構成することができる。

【0006】近年、開発されているDVD記録及び再生装置（DVDビデオレコーダ、DVDオーディオレコーダ、DVDビデオカメラ）に関して、一方で、DVDディスクに記憶されたサブタイトルのため、ハードウェア上の経費を伴なう上記サブピクチャーデコーディング装置を実装しなければならないという問題がある。また、他方で、今までのDVD規格には専用データパックが設けられていない付加的なタイトルデータを記録する必要があり、これを実現するため残されている唯一の可能性は、タイトルデータをビデオデータに対するデータパックにそのまま統合しなければならないという問題がある。これは、タイトルデータは常にビデオデータと連結され、タイトルデータをビデオピクチャーに挿入すべきか、或いは、タイトルデータをビデオピクチャーから消すべきかを選択する余地が残らないということを意味する。

【0007】

【発明が解決しようとする課題】本発明の目的は、特に、DVD記録及び再生用装置に、ビデオ及び／又はオーディオ信号に加えて、例えば、タイトル情報項目のような付加情報項目を別個に記憶させることができ、その結果として、回路構成に高い経費をかけることなく、目的の形式で付加情報項目をビデオピクチャーに挿入し、若しくは、ビデオピクチャーから除去することができるようになる、ビデオ及び／又はオーディオ信号の記録及び再生用の方法及び装置を提供することである。

【0008】

【課題を解決するための手段】本発明の目的は、請求項1及び11に記載された本発明の特徴によって実現される。請求項1に記載された新規な方法は、付加情報項目に対するデータをサブピクチャーユニットに変換し、サブピクチャーユニットは、記憶媒体上のビデオ及び／又はオーディオ信号に対するデータパックの他に、少なくとも1個のサブピクチャーデータパックに記録される。この解決法の場合、付加情報項目は従来のDVDプレーヤ用の規格に設けられたサブピクチャーユニット（SPU）に変換されるので、付加情報項目を記憶するため既存のDVD規格を変更する必要がない。また、本発明の更なる利点は、複数の異なる付加情報項目をこの形式で問題なく記録することができ、また、従来のDVDプレーヤは異なる言語のサブタイトルを選択するオプション機能を具備しているので、ユーザは再生中に挿入したい付加情報項目を選択することが可能である。

【0009】また、本発明によれば、少なくとも付加情報項目を再生するため使用されるサブピクチャーデコーディング装置は、サブタイトルのサブピクチャーユニットのデコーディングの間に使用されるサブピクチャーデコーディング装置と同じ装置であるため、付加情報項目の再生用回路に要する経費は非常に少ない。この場合、付加的な回路を設けなくても構わない。

【0010】請求項1に記載された方法の有利な展開及び改良は、従属請求項2乃至10に記載された構成によって実現され得る。請求項2には、サブピクチャーユニットの構造が詳細に定められている。特に、サブピクチャーユニットが付加情報項目の圧縮ビットマップにより構成されていることは、請求項2に記載されている。このようなサブピクチャーユニットの特性は、大容量のメ

モリ空間の節約を可能にさせる。かくして得られたメモリ空間は、交互に挿入され得る複数の異なる付加情報項目を並存的に記憶するため使用することができる。また、サブピクチャーユニットは、付加情報項目を表現する挿入用の制御命令をテーブルにより構成されることが請求項2に記載されている。このタイプの命令は、DVD規格により公知である。これは、挿入を構成するための別の可能性を与える。かくして、グラフィックシンボルを挿入することが可能であり、文字に対し異なるタイプのハイライトを使用することができる。したがって、付加情報項目はテキスト情報だけに限定されることがなく、特定のシンボル（記号）及びグラフィックス（図形）に関連付けることができる。

【0011】例えば、請求項3には、ランレングス符号化がビットマップの圧縮のため有利的に使用されることが記載されている。ランレングス符号化は簡単に実現される。請求項4よれば、付加情報項目は、オペレータによって入力され得、或いは、たとえば、再生時間、時刻又は日付を挿入する場合のように自動生成されることが記載されている。

【0012】請求項6に記載された非常に有利な構成は、予めランレングス符号化された文字を含むテーブルが設けられ、このテーブルから付加情報項目に必要な文字が選択され、サブピクチャーユニットの最終的なビットマップを形成するため合成される。請求項7には、各文字毎にテーブル内で行われる具体的な指定法が記載されている。

【0013】所望の付加情報項目が連続的に挿入される順序をユーザに指示させることができるプログラミングオプションは、本発明の方法により得られる利点である。付加情報項目を追加的に記録することができるビデオ及び／又はオーディオ信号の記録及び再生用の対応した記録及び再生装置は、請求項11に記載されている。

【0014】

【発明の実施の形態】以下、添付図面を参照して本発明の実施例を詳細に説明する。図1には、上記のDVD規格に準拠したビデオオブジェクトセット（VOBS）の構造が示されている。ビデオオブジェクトセットは、DVD規格に非常に厳密に記載されているDVDビデオディスクの論理データ構造の一部である。DVDディスクにはこの他のデータも記憶されているが、それらのデータは、ビデオ及びオーディオデータ、並びに、本発明に重要なサブピクチャーユニット用データを構成しないので、これ以上詳細な説明を行わない。したがって、他のデータユニットについては、DVD規格を参照のこと。図1に示されるように、ビデオブジェクトセット（VOBS）は、図1において、VOB_IDNiによって指定された多数のビデオオブジェクトにより構成される。各ビデオオブジェクトは、基本ストリームのグループを含むMPEGプログラムストリームにより構成される。この場合、ビデオ、オーディオ、サブピクチャー、PCI及びDSIの5種類の基本ストリームが存在する。各基本ストリームは多数のアクセスユニットを有する。かかるアクセスユニットは、例えば、グループオブピクチャー（GOP）、オーディオフレーム、サブピクチャーユニット、PCIパック及びDSIパックである。次に、ビデオオブジェクトはいわゆるセルに分割され、分割されたセルは図1に示されるように参照記号C_IDN1～C_IDNjで指定される。これらのセルはビデオオブジェクトユニット（VOBU）に分割される。VOBユニットはビデオオブジェクトの各部分である。これらの出現時間は、0．4乃至1秒間に亘って継続するので、各ビデオオブジェクトユニットは、例えば、12個のビデオピクチャーに関連した多数のGOPにより構成される。したがって、同じ量のデータがオーディオデータ及び／又はサブピクチャーユニットのデータのために存在する。ビデオオブジェクトユニットの構造の一例は、同様に図1に示されている。ナビゲーションパックNV_PCKは各ビデオオブジェクトユニットVOBUの先頭に配置される。ナビゲーションパックの後には、ビデオパックV_PCK及びオーディオパックA_PCKがサブピクチャーデータパックSP_PCKと共に可変順序で続く。これらのパックは、DVD規格では、“パック（Packs）”のように指定され、各パックは多数のサブパックを有し、サブパックはDVD規格では“パケット（Packets）”として指定される。しかし、この区別は、本発明の説明のためには副次的な重要性しかないので、以下の説明では、NV、V、A及びSPパックは、NV、V、A及びSPデータパックと称される。サブピクチャーユニット用のデータパック（SP_PCK）の構造は図2に示される。SP_PCK及びSP_PKTに対するヘッダ情報項目の後に詳細なSUB_Stream_IDが続けられ、これにより、32個の異なるサブピクチャーユニットを区別できるようになる。その後に、実際のサブピクチャーユニットのデータが続く。パックヘッダ及びパケットヘッダの両方の個別の詳細についてはDVD規格を参照のこと。

【0015】図3には、DVD規格に準拠したサブピクチャーユニット（SPU）の公知のデータフォーマットが概略的に示されている。同図には、サブピクチャーユニットのヘッダセクション（SPUH）のデータフィールド20と、サブピクチャーの第1のフィールドの圧縮画素データ（PXDTF）のデータフィールド21と、サブピクチャーの第2のフィールドの圧縮画素データ（PXDBF）のデータフィールド22と、表示制御コマンドシーケンステーブル（SP_DCSQT）のデータフィールド23とが示されている。いわゆる表示制御コマンドシーケンス（SP_DCSQ）はデータフィールド23に格納される。個々の表示制御コマンドについては、ここではこれ以上の説明を加えないが、公知のD

ＶＤ規格に非常に厳密に詳述されているので、本発明の開示のためにこの引用刊行物を簡単に参照する。図３には、表示制御コマンドシーケンス用のテーブルが複数の表示制御コマンドシーケンスＳＰ＿ＤＣＳＱ０乃至ＳＰ＿ＤＣＳＱ２を含むことが示されている。

【００１６】図４を参照するに、サブピクチャーユニットは、実際上、非常に大きいので、複数の連続したサブピクチャーデータパックＳＰ＿ＰＣＫｉ乃至ＳＰ＿ＰＣＫｊに記憶する必要がある。サブピクチャーユニットのデータフィールド２１及び２２内の画素データは、表示されたサブピクチャーのパターンを決定する。サブピクチャーの行の各画素毎に、２ビット幅のデータワードは、画素が文字画素若しくは背景画素の何れの画素であるか、画素が第１の方法若しくは第２の方法の何れの方法で強調されるべきかを指定する。これらの４通りの区別は２ビットを用いて行われ得る。本例の場合、２進数の値は具体的に以下の意味が与えられる。

００＝背景画素

０１＝文字画素

１０＝強調方法１を用いて表示された画素

１１＝強調方法２を用いて表示された画素

本例の場合、個別の画素データはそのままの形式で格納されるのではなく、圧縮形式で格納される。ランレングス符号化はこの目的のため使用される。具体的なランレングス符号化方法は以下に詳細に説明される。

【００１７】以下、図５及び６を参照して、本発明による記録及び再生装置の構造を説明する。本例の場合に、装置の記録機能のため不可欠であるとみなされる全ての基本構成要素は図５に示され、記録されたデータの再生のため必要とされる全ての基本構成要素は図６に示される。図５及び６において、同じ参照番号は同じ構成要素を示す。バッファ記憶装置４０は記録されるべきビデオデータを記憶する。ビデオデータは、例えば、ビデオカメラのような適当なデータソース、或いは、広帯域ケーブル、衛星受信器、若しくは、地上線受信用アンテナから発生する。着信データは、既にデジタル形式で表現されている。バッファ記憶装置４０で利用可能なビデオデータはＭＰＥＧ２ビデオエンコーダ回路４３によって処理され、このビデオデータはＭＰＥＧ２ビデオ標準に従って符号化される。このようにして符号化されたデータは、ビットストリームフォーマッティングユニット４９で利用できるようになり、このビットストリームフォーマッティングユニット４９において、このデータは、ＤＶＤ規格の論理データフォーマットに対応するビットストリームは出力で発生されるように組み立てられる。また、フォーマッティングユニット５０がさらに設けられ、データが再度フォーマットされ、その結果として、データは正確な物理的順序で組み立てられ、ＤＶＤディスク５１に記録するためそのまま使用することが可能である。データバッファ４１はオーディオデータ用のデータバッファである。データバッファ４１に格納されたデータはオーディオコーディング回路４４によって処理される。このオーディオコーディング回路４４は、例えば、ＭＰＥＧ方式オーディオ符号化回路、又は、ドルビーＡＣ３方式オーディオ符号化回路の何れでも構わない。このようにして発生されたデータは、次に、データフォーマッティングユニット４９で利用できるようになる。

【００１８】図５に示された装置は、キーボードユニット４７を更に有する。キーボードユニット４７はマイクロコントローラ４６に接続される。ユーザはキーボードユニット４７を介して入力を実行することができる。特に、ユーザは、例えば、同時に記録されるべき所望のタイトルを入力することができる。勿論、他の所望の入力がキーボードユニット４７を用いて同様に実現される。マイクロコントローラ４６において、入力されたデータは、論理的に順序付けられ、バッファ記憶装置４２に転送される。バッファ記憶装置４２に格納されたデータは、サブピクチャーコーディングユニット４５によって修正される。入力されたデータに対し、サブピクチャーユニットがサブピクチャーコーディングユニット４５で生成される。サブピクチャーユニットは、次に、データフォーマッティングユニット４９に転送される。サブピクチャーユニットはデータフォーマッティングユニット４９に転送される。ビデオデータパック、オーディオデータパック、及び、サブピクチャーデータパック用のデータパックの定型化は、好ましくは、データフォーマッティングユニット４９において行われる。

【００１９】実時間クロック４８はマイクロコントローラ４６に付加的に接続される。時刻データ及び日付細目は、実時間クロックを用いて通知され得る。上記データ及び細目は、マイクロコントローラ４６によって変換され、特殊挿入（サブピクチャー挿入）のため使用される。個々までの説明では、記録されたデータが多数の誤りプロテクションによって保護されることについて記載していない。この誤りプロテクションによる保護は、フォーマッティングユニット５０によって行われる。

【００２０】或いは、サブピクチャー符号化ユニット４５は、マイクロコントローラ４５が充分に強力であるならば、マイクロコントローラ４６に統合してもよい。以下、図６に示されたブロック構成図を参照して、記録されたデータの再生中に使用される基本構成要素について説明する。シリアルデータ入力５８は、ビデオデータ及びオーディオデータ、並びに、サブピクチャー用データを含むビットストリームが与えられる。このデータは光学式記憶ディスクＤＶＤ５１によって供給される。着信データは、最初に、訂正ユニット６０において誤り検出及び誤り訂正を施される。データは次に分離回路６１に伝達され、データ中に混合されているビデオデータ、オーディオデータ及びサブピクチャーデータが分離され、

エントリーから正確に得られるようにプログラムステップ１０６で改めて計算されるべきことに注意する必要がある。プログラムステップ１０４において最後の文字に達したことが判定された場合、プログラムステップ１１０において、画素行に対するカウンタは２行ずつ進められ（ＲＯＷ：＝ＲＯＷ＋２）、文字数に対するカウンタは１にリセットされる（ＣＨＡＲＮ：＝１）。また、残りの背景画素の個数は判定され、関連したランレングス符号がこのプログラムステップ１１０で決定される。最後に、画素行に対し生成されたランレングス符号の全体が整数個のバイトに収容され得るかどうかが判定される。収容できない場合、本例では、符号化された画素行は４個の零ビット（１ニブル）によって対応的に長さ制限されるので、符号化された画素列はバイト境界で終了する。かくして得られた画素行のデータは、対応して記憶される。全ての画素行はこのように順番に生成される。プログラムステップ１０３において、画素行の数ＲＯＷが７よりも大きいことが確定された場合、処理はプログラムステップ１１１（ＷＴＥＰＬ）に進み、文字の下側にある背景画素を含む空の画素行は、更にランレングス符号化を施される。プログラムはプログラムステップ１１２（Ｅ）で終了する。

【００２６】図７に示されたプログラムは１フィールド（第１又は第２フィールド）の圧縮ビットマップを作成する機能も行うことに注意する必要がある。他のフィールドに対する圧縮ビットマップを作成するため、異なる開始値が運転中変数のため使用される対応するプログラムが適用可能である。画素行を処理し、その画素値をランレングス符号に与えるため必要なプログラムステップは、図７の破線で囲まれた領域に示される。これは、図７の下方部に示されたサブピクチャーの例によって示される。

【００２７】第１及び第２のフィールドに対する圧縮ビットマップの生成後、完全なサブピクチャーユニットＳＰＵは図８のプログラムに従って発生される。このプログラムは、ステップ１２０（Ｓ）から始まり、プログラムステップ１２１において、第１のフィールドの圧縮ビットマップを作成し（ＧＥＮ　ＣＯＭＰ　ＢＩＴＭＡＰ　ＴＦ）、ステップ１２２において、第２のフィールドの圧縮ビットマップを作成する（ＧＥＮ　ＣＯＭＰ　ＢＩＴＭＡＰ　ＢＦ）。これらのプログラムステップにおいて、何れの場合もプログラムは図７に示されたフローに従って処理される。続いて、ステップ１２３において、必要な表示制御コマンドシーケンスが生成される（ＧＥＮ　ＳＰ＿ＤＳＱ０，．．．）。サブピクチャーのビデオピクチャーへの挿入の開始時点及び終了時点、並びに、例えば、画面上の挿入位置は、対応したコマンドを用いて定義される。これらのコマンド及びアプリケーションはＤＶＤ規格により定義されるので、ここで、ＤＶＤ規格を参照する。プログラムステップ１２４において、この点までに編成された表示制御コマンドシーケンステーブルの長さは補助的に調べられる。ＤＶＤ規格に準拠して、この長さはサブピクチャーユニットの全体のサイズの半分以下でなければならない。この規則が充足されない場合、規則が満たされるように充填用ビットが付加される（ＧＥＮ　ＳＴＵＦＦ）。

【００２８】最後に、プログラムステップ１２５において、サブピクチャーユニットの導入ヘッダセクションＳＰＵＨがＤＶＤ規格の規則に従って作成される（ＧＥＮ　ＳＰＵＨ）。プログラムはプログラムステップ１２６で終了する（Ｅ）。対応するサブピクチャーデータパックは、図９に示された３番目のフローチャートを用いて、サブピクチャーユニットに対しこの点までに作成されたデータから生成される。このプログラムセクションはプログラムステップ１３０から始まる（Ｓ）。サブピクチャーデータパックのヘッダセクションはプログラムステップ１３１によって作成される（ＧＥＮ　ＰＣＫＨ）。ＤＶＤ規格による“パックヘッダ”はこの場合に関係する。この機能は、ＤＶＤ規格に適切に説明されているので、これ以上詳細に説明する必要はないであろう。プログラムステップ１３２において、サブピクチャーユニットは、図８に関して説明した方法で作成される。データはパックの作成が終わるまでバッファ記憶される（ＧＥＮ　ＳＰＵ）。パケットのヘッダセクション（“パケットヘッダ”）はプログラムステップ１３３で作成される（ＧＥＮ　ＰＫＴＨ）。この場合、サブピクチャーユニットは単一のパケットに詰め込むことが可能である。或いは、複数のパケットを作成する必要がある。最後に、全ての部分は合成され、完成したデータパックが発生され、記録のため利用できるようになる。これはプログラムステップ１３４で行われる（ＡＳＳ　ＳＰ＿ＰＣＫ）。プログラムはプログラムステップ１３５で終了する（Ｅ）。

【００２９】実時間サブピクチャーデータパック発生の機能をよりよく理解するため、図１０乃至１８には、ＡＮＳＩ準拠Ｃプログラミング言語で記述されたプログラムリストが掲載されている。このプログラムリストは、図７乃至９を参照して説明した３つのプログラムセクションを含む。ランレングス符号化文字のテーブルはプログラムリストの最初の部分（セクションＡ）に掲載される。プログラムリストを簡単化するため、１０個の数字０乃至９、スペース文字、及び、コロンがテーブルに列挙される。また、４×７ドット形のドットマトリックスが各文字毎に使用される。既にランレングス符号化された文字を含むテーブルを使用することにより、このプログラムは非常に高速に動作することが可能であり、したがって、ここで要求される実時間性能に特に適している。大きいドットマトリックス、例えば、１０×１６画素のドットマトリックスが１文字毎に使用された４×７画素のドットマトリックスの代わりに使用される場合、

ランレングス符号化の複雑さが増大するので、全ての文字を予め符号化する利点が生じる。この場合、一般的に、文字を用いてタイトル表示が実現できるように、文字セットが使用される。その上、画面上でよりよく識別できるように非常に大きい文字を使用する必要がある。また、大量のメモリ容量を節約するビットマップ生成は、テーブルが先行（左側）背景画素の個数及び後続（右側）背景画素の個数を文字の各画素行のランレングスとして含むので、テーブルの構造に依存する可能性があるとしても非常に高速である。これにより、ある画素から次の画素に画素行内で、プログラミングに関して簡単であり、プロセッサに関して高速であり、かつ、同時にメモリに関して最適化されたランレングス符号化を行える。その結果として、大多数のアプリケーションの場合に、このようなサブピクチャーデータパックの経済的なコーディングが行われるので、困難性を伴うことなく単一セクター内にこのようなデータパックのための空間が設けられる。したがって、かくして生成されたサブピクチャーユニットに対しＤＶＤディスクに必要とされるメモリは最小限に抑えられる。

【００３０】プログラムリスト中、７画素行がテーブルの各文字毎に記述される。テーブル中の各画素行の説明は、厳密に４個のエントリーを含む。

１．ニブル内の各画素行のランレングス符号の長さ

２．画素行内の文字の最初の文字画素までの背景画素の個数

３．画素行内の文字の最後の文字画素以降の背景画素の個数

４．先行及び後続背景画素を伴わない画素列のランレングス符号

上記の文字“０”のためのテーブルエントリーは一例として記載されている。本例の場合、以下の記法

“．”＝背景画素

“＋”＝文字画素

が使用される。

第１行：　．＋＋．

第２行：　＋．．＋

第３行：　＋．．＋

第４行：　＋．．＋

第５行：　＋．．＋

第６行：　＋．．＋

第７行：　．＋＋．

例えば、以下の４通りのテーブルエントリーは第２行（“＋．．＋”）に対し生成される。

１．３個のニブル（ランレングス符号は以下の４に記載されるように５８５ｈである）。

２．最初の“＋”画素の左側の０背景画素

３．最後の“＋”画素の右側の０背景画素

４．５８５ｈ（夫々のランレングス１（２進数：０１）及び文字画素コード０１（２進数）を有し、これにより、文字２進数０１０１＝１０進数表現の５と、２個の内部背景画素に対するランレングスコード８（２進数：１０００）の２個文字画素により構成され、文字画素＋２個の背景画素＋１個の文字画素＝５＋８＋５によって、５８５ｈが得られる）。

【００３１】また、例えば、第７行（“．＋＋．”）に対し以下の４通りのテーブルエントリが生成される。

１．１個のニブル（ランレングスコードは、以下の４に記載されているように１個の９である）。

２．最初の文字画素の左側の１背景画素

３．最後の文字画素の右側の１背景画素

４．９（ランレングス２（２進数：１０）と文字画素コード０１（２進数）とにより構成され、これにより、２進数１００１＝１０進数９が形成される）。

【００３２】サブピクチャーユニットの圧縮ビットマップを生成する機能はプログラムリストのなかでセクションＢによって示された部分に掲載される。サブピクチャーユニットを生成するプログラムのソーステキストは、プログラムリストのセクションＣに掲載される。サブピクチャーデータパックを発生させる機能のためのプログラムのソーステキストは、リストの最後のセクションＤに掲載されている。プログラムリストには対応したコメントが付記されているので、プログラムの個々のセクションは容易に識別され得る。例をできる限り簡単に保つため、殆どのエラー処理はリスト上で省略されている。

【００３３】関数呼び出しの３通りの例が図１９に示されている。対応したプログラムリストは、ＡＮＳＩ準拠Ｃ言語で記述されている。第１の例において、テキスト“０１：２３：４５：６７：８９”はテキスト文字列として定義される。プログラムは、関連したサブピクチャーユニットを自動的に生成する。第２の例の場合、テキスト“０　１　２　３　４　５”はテキスト文字列として転送される。最後に、第３の例の場合、対応したストップクロックの現在再生時間はテキスト文字列として規定される。一例として、再生時間“００：０１：４５”が記載されている。

【００３４】関数呼び出しの第４の例は、図２０及び２１に記載されている。結果的に得られたサブピクチャーデータパックの１６進数は図２０の下側に掲載されている。最後に、パック内に存在するサブピクチャーユニットに対する１６進数及び対応したコメントは、その後に掲載されている。上記のサブピクチャーユニットに従って表示されたサブピクチャーを象徴する簡単なグラフィックがその後に掲載されている。

【００３５】

【実施例】図２２の（ａ）乃至（ｃ）と、図２３の（ｄ）乃至（ｆ）と、図２４の（ｇ）乃至（ｉ）は、種々のアプリケーションの可能性を示す図である。図２２の（ａ）によれば、画面上の表示にはサブピクチャーが挿入されていない。したがって、サブピクチャーデコー

ディングユニットはピクチャーを出力しない。図２２の（ｂ）には、例えば、記録日付がピクチャー中に挿入される。図２２の（ｃ）では、記録時刻がピクチャーに挿入される。

【００３６】図２３の（ｄ）において、日付及び時刻がサブピクチャーとして同時に挿入される。しかし、この場合、２個の異なるサブピクチャーユニットを符号化する必要はなく、共通サブピクチャーユニットは２個の情報項目を含む。図２３の（ｅ）において、現在再生時間はピクチャーに挿入される。図２３の（ｆ）において、ユーザによって規定された特殊テキストがピクチャーに挿入される。

【００３７】図２４の（ｇ）では、記録ビデオピクチャーのタイトルがピクチャーに挿入される。図２４の（ｈ）において、一方で、再生時間に関係する情報を含み、他方で、記録機能の作動中の指示を含むサブピクチャーがビデオピクチャーに挿入される。この挿入は、ＤＶＤビデオカメラの場合に、例えば、ビューファインダーへ流用することが可能であり、その結果として、挿入項目は記録中にオペレータへ情報を知らせるため使用される。図２４の（ｉ）には、特殊グラフィック挿入を実現する一例が示されている。この例の場合、サブピクチャーユニットは、双眼鏡を通して見える視界を表すマスクを含む。

【００３８】秒単位で正確な再生時間が挿入される例の場合、新しいサブピクチャーユニットが発生され、１秒毎に記憶させる必要がある。このアプリケーションは、最もメモリ集約的なアプリケーションである。その理由は、サブピクチャーユニットが、本例の場合に（１秒毎に）非常に素早く変更される必要があるからである。毎秒５Ｍビットの記録レートを想定し、各サブピクチャーユニットがＤＶＤの１セクターを正確に占有する場合を想定すると、残りのビデオ及びオーディオデータと、それ以外のサブピクチャーユニットとに対する約３０５個のセクターは、再生時間サブピクチャーユニットのための二つの記録されたセクターの間に収まる。これは、最もメモリ集約性の高いサブピクチャーユニットが記録媒体の総容量の高々０．３％しか占有しないことを意味する。しかし、このメモリ要求量は、適切な圧縮によって、さらに、０．１％未満まで減少させることができる。適当な圧縮として、例えば、５秒間の表示を完全に収容する記録時間サブピクチャーユニットを用いることによって、すなわち、記録時間サブピクチャーユニットは、５種類の表示に対するビットマップデータを収容し、１秒毎に表示の内容を自動的に切り換えることにより、メモリ要求量が減少される。

【００３９】ユーザが所望の付加情報項目が連続的に挿入される順序を規定できるプログラミング上の選択の幅は、サブピクチャーデコーディングユニットの記憶装置内のテーブルに、その時点で選択されるべきサブピクチャーユニット（すなわち、サブストリームｉｄ）が入れられ、そのサブピクチャーユニットが次に処理されるように、実現することが可能である。

【００４０】本発明は、ＤＶＤビデオレコーダ及び／又はＤＶＤ音楽レコーダに利用することができる。音楽レコーダの場合、タイトル細目等は画面を介して情報として出力され得る。本発明は、ＤＶＤ記録及び再生装置におけるアプリケーションのため広範に使用することが可能である。現在開発中のＤＶＤのＲＡＭ装置及びＤＶＤ記録装置、並びに、ＤＶＤビデオカメラ及びＤＶＤビデオレコーダは、特に、指摘される。しかし、アプリケーションの可能性は、これらの例に限定されない。サブピクチャーユニットが他の記録装置の別の世代に使用されるとしても、アプリケーションの可能性は妨げられない。

【図面の簡単な説明】

【図１】ビデオデータパケット、オーディオデータパケット及びサブピクチャーデータパケットに分割されたＤＶＤの記録されたデータストリームを表す図である。

【図２】サブピクチャーデータパックの構造を示す図である。

【図３】サブピクチャーユニットの構造を示す図である。

【図４】多数のサブピクチャーデータパケットに分割されたサブピクチャーユニットを表す図である。

【図５】記録及び再生装置の記録部に関係したブロック図である。

【図６】記録及び再生装置の再生部に関係したブロック図である。

【図７】サブピクチャーユニットの圧縮ビットマップを発生させるプログラムのフローチャートである。

【図８】付加情報項目をサブピクチャーユニットに変換する処理のフローチャートである。

【図９】付加情報項目からサブピクチャーデータパケットを発生させるプログラムの概略的なフローチャートである。

【図１０】実時間サブピクチャーデータパケット発生のプログラムの一例を示すプログラムリストである。

【図１１】実時間サブピクチャーデータパケット発生のプログラムの一例を示すプログラムリストである。

【図１２】実時間サブピクチャーデータパケット発生のプログラムの一例を示すプログラムリストである。

【図１３】実時間サブピクチャーデータパケット発生のプログラムの一例を示すプログラムリストである。

【図１４】実時間サブピクチャーデータパケット発生のプログラムの一例を示すプログラムリストである。

【図１５】実時間サブピクチャーデータパケット発生のプログラムの一例を示すプログラムリストである。

【図１６】実時間サブピクチャーデータパケット発生のプログラムの一例を示すプログラムリストである。

17

【図１７】実時間サブピクチャーデータパケット発生のプログラムの一例を示すプログラムリストである。

【図１８】実時間サブピクチャーデータパケット発生のプログラムの一例を示すプログラムリストである。

【図１９】図１０乃至１８に示されたプログラムに関係した関数呼び出しの３通りの例を示す図である。

【図２０】第４の詳細な関数呼び出しの例を示す図である。

18

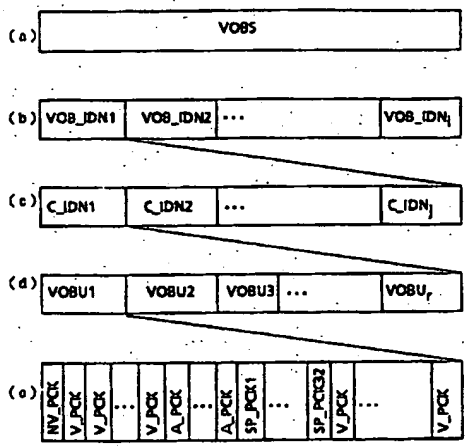
【図２１】第４の詳細な関数呼び出しの例を示す図である。

【図２２】（ａ）、（ｂ）及び（ｃ）は付加情報挿入のアプリケーション例を示す図である。

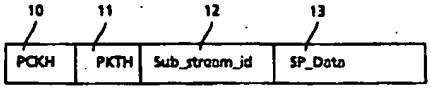
【図２３】（ｄ）、（ｅ）及び（ｆ）は付加情報挿入のアプリケーション例を示す図である。

【図２４】（ｇ）、（ｈ）及び（ｉ）は付加情報挿入のアプリケーション例を示す図である。

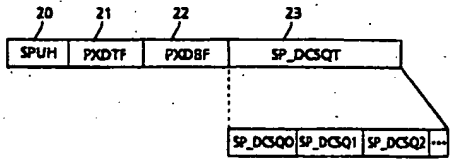
【図1】



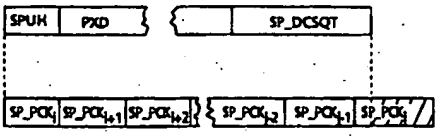
【図2】



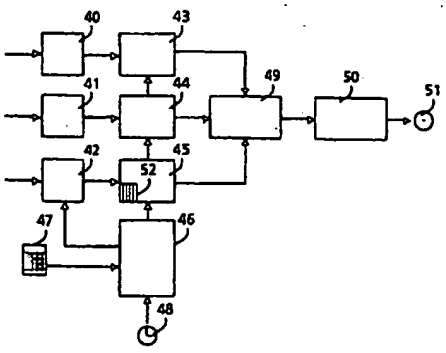
【図3】



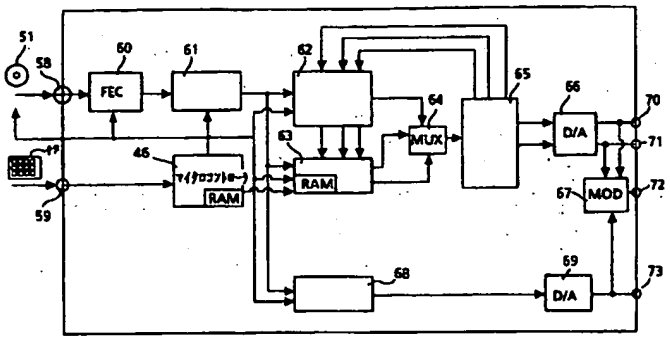
【図4】



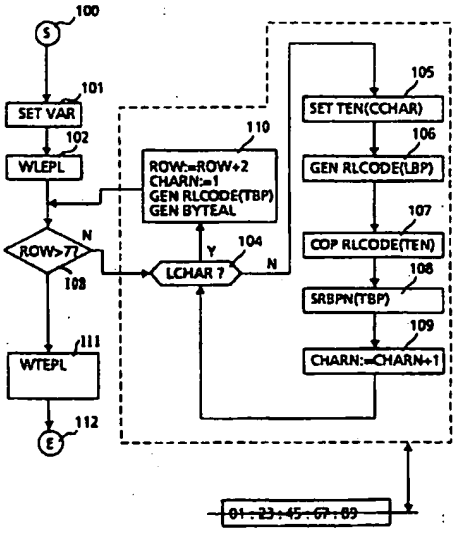
【図5】



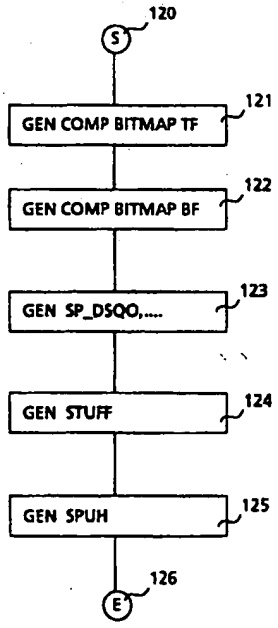
【図6】


51
58
60
FEC
61
62
65
66
70
64
MUX
D/A
71
63
RAM
49
46
RAM
59
67
MOD
72
68
69
D/A
73

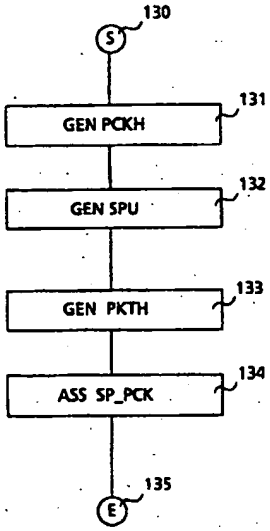
【図7】


100
S
101
SET VAR
102
WLEPL
110
ROW:=ROW+2
CHARN:=1
GEN RLCODE(TBP)
GEN BYTEAL
N
ROW>77
108
Y
104
LCHAR ?
N
105
SET TEN(CCHAR)
106
GEN RLCODE(LBP)
107
COP RLCODE(TEN)
108
SRBPN(TBP)
109
CHARN:=CHARN+1
111
WTEPL
E
112
01・23・45・67・89

【図8】


120
S
GEN COMP BITMAP TF
121
GEN COMP BITMAP BF
122
GEN SP_DSQO,....
123
GEN STUFF
124
GEN SPUH
125
E
126

【図9】



【図11】

```
{1, 1, 1, {0x90,  0   } }, /*   .11.  */
{3, 0, 1, {0x54, 0x50 } }, /*   1.1.  */
{1, 2, 1, {0x50,  0   } }, /*    1.   */
{1, 2, 1, {0x50,  0   } }, /*   ..1.  */
{1, 2, 1, {0x50,  0   } }, /*   ..1.  */
{1, 1, 0, {0xD0,  0   } } /*   .111  */
}, {/*** 2 ***/
{1, 1, 1, {0x90,  0   } }, /*   .22.  */
{3, 0, 0, {0x58, 0x50 } }, /*   2.2.  */
{1, 3, 0, {0x50,  0   } }, /*   ...2  */
{1, 2, 1, {0x50,  0   } }, /*   .2.   */
{1, 1, 2, {0x50,  0   } }, /*   .2.   */
{1, 0, 3, {0x50,  0   } }, /*   2...  */
{2, 0, 0, {0x11,  0   } } /*   2222  */
}, {/*** 3 ***/
{1, 1, 1, {0x90,  0   } }, /*   .33.  */
{3, 0, 0, {0x58, 0x50 } }, /*   3..3  */
{1, 3, 0, {0x50,  0   } }, /*   ...3  */
{1, 2, 1, {0x50,  0   } }, /*   ..3.  */
{1, 3, 0, {0x50,  0   } }, /*   ...3  */
{3, 0, 0, {0x58, 0x50 } }, /*   3..3  */
{1, 1, 1, {0x90,  0   } } /*   .33.  */
}, {/*** 4 ***/
{1, 0, 3, {0x50,  0   } }, /*   4...  */
{1, 0, 3, {0x50,  0   } }, /*   4...  */
{3, 0, 1, {0x54, 0x50 } }, /*   4.4.  */
{3, 0, 1, {0x54, 0x50 } }, /*   4.4.  */
{2, 0, 0, {0x11,  0   } }, /*   4444  */
{1, 2, 1, {0x50,  0   } }, /*   ..4.  */
{1, 2, 1, {0x50,  0   } } /*   ..4.  */
}, {/*** 5 ***/
{2, 0, 0, {0x11,  0   } }, /*   5555  */
{1, 0, 3, {0x50,  0   } }, /*   5...  */
{1, 0, 3, {0x50,  0   } }, /*   5...  */
{1, 0, 1, {0xD0,  0   } }, /*   555   */
{1, 3, 0, {0x50,  0   } }, /*   ...5  */
{3, 0, 0, {0x58, 0x50 } }, /*   5..5  */
{1, 1, 1, {0x90,  0   } } /*   .55.  */
}, {/*** 6 ***/
{1, 1, 1, {0x90,  0   } }, /*   .66.  */
{1, 0, 3, {0x50,  0   } }, /*   6...  */
{1, 0, 3, {0x50,  0   } }, /*   6...  */
{1, 0, 1, {0xD0,  0   } }, /*   666.  */
{3, 0, 0, {0x58, 0x50 } }, /*   6..6  */
{3, 0, 0, {0x58, 0x50 } }, /*   6..6  */
{1, 1, 1, {0x90,  0   } } /*   .66.  */
}, {/*** 7 ***/
{2, 0, 0, {0x11,  0   } }, /*   7777  */
{3, 0, 0, {0x58, 0x50 } }, /*   7..7  */
{1, 3, 0, {0x50,  0   } }, /*   ...7  */
{1, 2, 1, {0x50,  0   } }, /*   .7.   */
{1, 1, 2, {0x50,  0   } }, /*   .7.   */
{1, 1, 2, {0x50,  0   } }, /*   .7.   */
{1, 1, 2, {0x50,  0   } } /*   .7.   */
}, {/*** 8 ***/
{1, 1, 1, {0x90,  0   } }, /*   .88.  */
{3, 0, 0, {0x58, 0x50 } }, /*   8..8  */
{3, 0, 0, {0x58, 0x50 } }, /*   8..8  */
{1, 1, 1, {0x90,  0   } }, /*   .88.  */
{3, 0, 0, {0x58, 0x50 } }, /*   8..8  */
{3, 0, 0, {0x58, 0x50 } } /*   8..8  */
```

A

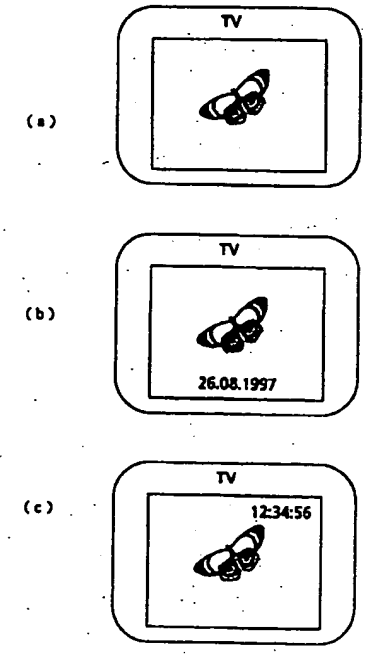
【図１０】

```
/* ----------------------------------------------------------------------------
   Copyright (c) 1998, This software is the property of Thomson
   multimedia, and shall not he reproduced, copied, or distributed
   without written permission.
---------------------------------------------------------------------------- */

/* ----------------------------------------------------------------------------
   AUTHOR          Marco Winter
   DATE CREATED    17-MAR-1998
   PROJECT :       DVD, realtime sub-picture pack generation
   DESCRIPTION :
     Realtime generating of sub-picture packs

---------------------------------------------------------------------------- */

/* ********************** INCLUDES: ********************************************/
#include <string.h>

/* ***************** ******* DEFINES: ********************** ********** *********/
#define N_ROWS 7                                /* pixel heigth of a character */
#def ine SPACE_BETWEEN_2_CHARACTERS 2    /* no. of pels between 2 characters */

/* ******************* ******* ******* TYPE DEFINITIONS: ************************/
typedef  unsigned  char   Uint8;
typedef  unsigned  short  Uint16
typedef  unsigned  long   Uint32;
typedef            char   Int8;
typedef            short  Int16;
typedef            long   Int32;

/* ************** ************************** VARIABLES: **********************/

/* control variables for bit manipulation:    */
static Uint8     tmpbuf =0;
static Uint32    tmppos =0;
static size_t    tmplen =0;

/* a pointer inside the compressed bitmap:   */
static Uint8    *bitmap;

/* struct for the run-length code of one row of a character:   */
struct CharacterRow {
  Uint32 nNibbel,                /* length of the run-length code in nibbel */
         nBeginningPel,          /* leading background pel */
         nRemainingPel;          /* trailing background pel */
  Uint8  runLengthCode [2];      /* the run-length code itself */
};

/* run-length code of the characters:   */
const struct CharacterRow characterSet[ ] [N_ROWS]  = {
 {/*** 0 ***/
   { 1, 1, 1, { 0x90,    0 } }, /* .00. */
   { 3, 0, 0, { 0x58, 0x50 } }, /* 0..0 */
   { 3, 0, 0, { 0x58, 0x50 } }, /* 0..0 */
   { 3, 0, 0, { 0x58, 0x50 } }, /* 0..0 */
   { 3, 0, 0, { 0x58, 0x50 } }, /* 0..0 */
   { 3, 0, 0, { 0x58, 0x50 } }, /* 0..0 */
   { 1, 1, 1, { 0x90,    0 } }  /* .00. */
 }, {/*** 1 ***/
   { 1, 2, 1, { 0x50,    0 } },   /* ..1. */
```

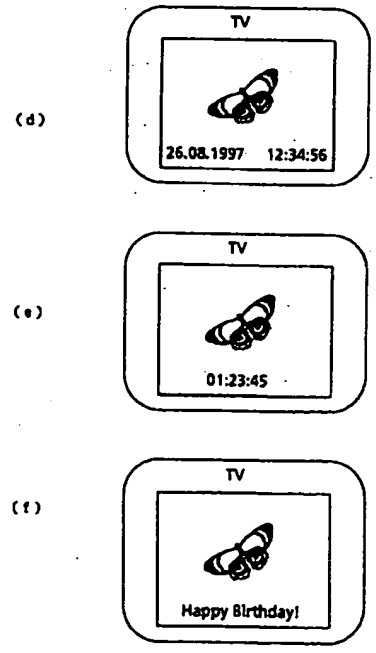
【図12】

```
  { 1, 1, 1,  {0x90,      0  }},  /*    .88.    */
},{ /*** 9 ***/
  { 1, 1, 1,  {0x90,      0  }},  /*    .99.    */
  { 3, 0, 0,  {0x58,   0x50  }},  /*    9..9    */
  { 3, 0, 0,  {0x58,   0x50  }},  /*    9..9    */
  { 1, 1, 0,  {0xD0,      0  }},  /*    .99.    */
  { 1, 3, 0,  {0x50,      0  }},  /*    ...9    */
  { 1, 3, 0,  {0x50,      0  }},  /*    ...9    */
  { 1, 1, 1,  {0x90,      0  }},  /*    .99.    */
},{ /*** : ***/
  { 0, 4, 0,  {   0,      0  }},  /*    ....    */
  { 0, 4, 0,  {   0,      0  }},  /*    ....    */
  { 1, 1, 2,  {0x50,      0  }},  /*    :.      */
  { 0, 4, 0,  {   0,      0  }},  /*    ....    */
  { 1, 1, 2,  {0x50,      0  }},  /*    :.      */
  { 0, 4, 0,  {   0,      0  }},  /*    ....    */
  { 0, 4, 0,  {   0,      0  }},  /*    ....    */
},{ /*** SPACE ***/
  { 0, 4, 0,  {   0,      0  }},
  { 0, 4, 0,  {   0,      0  }},
  { 0, 4, 0,  {   0,      0  }},
  { 0, 4, 0,  {   0,      0  }},
  { 0, 4, 0,  {   0,      0  }},
  { 0, 4, 0,  {   0,      0  }},
  { 0, 4, 0,  {   0,      0  }}
 }
};

/* generates byte alignment for the compressed bitmap of a SP line:  */
#define putAlign( )  if( !tmppos); else putHBit( (Uint32) 0, 8-tmppos)

/* puts the <n> most significant bits of the 32 bit value <b>
 * into buffer <bitmap>:  */
static void putBit( Uint32 b,  Int32 n) {

 */ significant  bits in <b> are on the left side of Uint32 word */

 while( --n>=0 )  {
  tmppos++;
  tmpbuf <<= 1;
  if( b&(unsigned long)0x80000000 )  tmpbuf++;
  b <<= 1;
  if(  tmppos == 8 )  {
    *bitmap++ = tmpbuf;
    tmppos = 0;
    tmplen++;
   }
 }

}/* putBit( ) */

/* puts the <n> most significant bits of the first 8 bits of
 * value <b> into buffer <bitmap>:  */
static void putHBit( Uint32 b,  Int32 n )  {
 putBit( b<<24,          /* significant bits to left long word limit */
        n );
}/* putHBit( )  */
/* puts the <n> least significant bits of the 32 bit value <b>
 * into buffer <bitmap>:  */
static void putLBit( Uint32 b,  Int32 n )  {
 putBit( b<<(32-n),      /* significant bits to left long word limit */


        n );
 }/* putLBit( )  */
```

【図22】



【図23】



【図１３】

```
/*****************************************************************
*
*    Extern Function: makeCompressedBitmap ( )
*
*    Encodes characters from a text string to run-length encoded data
*    Input: pointer to an already allocated buf fer for the compressed bitmap
*           and a text string which shall be encoded as SP
*    Output: the run-length coded (=compressed) bitmap
*
*    Return value: byte length of the by this function just generated bitmap
*
*****************************************************************/
extern Uint2 makeCompressedBitmap (
        Uint8   *startofBitmap,  /* OUT: destination of the compressed data */
        Uint32   topField,       /* 0: only bottom field;  1: only top field */
        char     *completeText,  /* text which shall be encoded as SP */
        Uint32   offsetLeft,     /* background pixel offset left */
        Uint32   offsetRight,    /* background pixel offset right */
        Uint32   offsetUp,       /* background pixel offset up */
        Uint32   offsetDown,     /* background pixel offset down */
        Uint32 *spWide,          /* OUT: will be set to the horizontal
                                  *      pixel size of the whole SP */
        Uint32 *spHight          /* OUT: will be set to the vertical
                                        pixel size of the whole SP */
        ) {
 char  *text;                    /* pointer to move inside <completeText> */
 Uint32 lastNBackgroundPel;      /* count remaining background pel */
 Uint32 row;                     /* geometrical row inside the SP */
 Uint32 character;               /* character element no. of the char. set */
 Uint32 n;                       /* bit counter for the run-length codes */

 /* horizontal size of the SP in pel:   */
 *spWide =  (offsetLeft +
        strlen( completeText )   * (4+SPACE_BETWEEN_2_CHARACTERS) -
        SPACE-BETWEEN_2_CHARACTERS +
        offsetRight) ;
 /* vertical size of the WHOLE   (not only the field ! ) SP in pel: */
 *spHight = offsetUp + N_ROWS + offsetDown;

 /* init. pointer to move inside the compressed data area:   */
 bitmap = startOfBitmap;

 /* upper background pixel offset of the SP   */
 for( row=topField; row<offsetUp; row += 2 )  {
   *bitmap++ = 0;
   *bitmap++ = 0;


 /* encoding the actual text:  */
 for( row=(offsetUp+topField) &1;  row<N_ROWS      row += 2 ) {
   text = completeText;                          /* init text ptr */
   lastNBackgroundPel = offsetLeft;              /* left background pixel
                                                  * offset of the SP */
   /* encode line <row> of the whole text */
   while( *text ! = '\0' ) { /* encode line <row> of char. *text */

     if( *text >='0'     && *text <= '9' )        /* it's a digit */
      character =   (Uint32)*text - '0' );
```

【図１４】

```
else                                        /* it's a special char.   */
 switch( *text ) {
 case ' ':  character = 10; break;
 default:  character = 11; break;           /* use space as default */

/* get leading background pixel: */
n = lastNBackgroundPel + characterSet[ character ][ row ].nBeginningPel;

/* when there are pattern pixel for this row then encode those:   */
if( characterSet[ character ][ row ].nNibbel )  {

  /* encode leading background pixel (if any):    */
  while ( n ) {   /* generate run-length code     */
    if ( n<16)
    if ( n<4)
      putLBit( n, 2 );   /* put the 2 LSB of n onto bitmap */
    else
      putLBit( n, 6 );
   else
      if( n<64,)
      putLBit( n, 10 );
    else if( n<256)
      putLBit( n, 14 );
    else
      putLBit( 255, 14);
   putHBit( 0, 2 );
   if( n >= 256 ) n -= 255;
   else    n = 0;
  }

  /* insert preencoded character run-length code:    */
  for( n=0; n<characterSet[ character ][ row ].nNibbel; n++ ) {
   if( n&1 )    /* odd */
    putLBit( characterSet[ character ][ row ].runLengthcode[n>>1],    4 );
   else /* even */
    putHBit( characterSet[ character ][ row ].runLengthcode[n>>1],    4 );
  }
  /* prepare encoding of trailing background pixel    (if any): */
  lastNBackgroundPel = characterSet[ character ][ row ].nRemainingPel;

 } else {/* there are no pattern pixel to encode */

  /* prepare encoding of background pixel (if any):    */

  lastNBackgroundPel = n + characterSet[ character ][ row ].nRemainingPel;
 }

 /* insert space between 2 characters: */
 lastNBackgroundPel += SPACE_BETWEEN_2_CHARACTERS;
 text++; /* next character */
} /* while */

/* encode the remaining background pels of the current SP row:    */
n = lastNBackgroundPel + offsetRight - SPACE_BETWEEN_2_CHARACTERS;
if( n ){
 if( n<16 )
  if( n<4)
   putLBit( n, 2 );  /* put the 2 LSB of n onto bitmap */
  else
```

B

【図１５】

```
    putLBit( n, 6 );
  else
   if( n<64 )
    putLBit( n, 10 );
   else if( n<256)
    putLBit( n, 14 );
   else
    putLBit( 0, 14);
  putHBit( 0, 2 );


 /* add nibbel stuffing in order to get the SP line byte aligned */
 putAlign( );
} /* for row */

/* lower background pixel offset of the SP: */
for( row=(offsetUp+N_ROWS+topField)&1; row<offsetDown; row += 2 ) {
 *bitmap++ = 0;
 *bitmap++ = 0;
}

/* return the byte length of the just encoded compressed data */
return (Uint32) (bitmap - startOfBitmap);

} /* makeCompressedBitmap( ) */

/*****************************************************************************
 *
 *   Extern Function: makeSPU( )
 *
 *   Encodes characters from a text string to complete SPU
 *   Input: pointer to an already allocated buffer for the complete SPU
 *       and a text string which shall be encoded as SP
 *   Output: the encoded SPU
 *
 *   Return value: length of the whole SPU
 *
 *****************************************************************************/
extern Uint32 makeSPU ( )
        Uint8   ntsc,                   /* 0: 625/50; 1: 525/60 */
        Uint8  *spuBuffer,              /* OUT: already allocated buf fer for the SPU */
        char   *text,                   /* this text shall be encoded as SP */
        Uint32  durationInFrames,       /* after this no. of frames
                                         * the SP shall disappear */
        Uint32 starty                   /* vertical start position of the SP */
        ) {
 Uint32
  spWide,       /* horizontal size of SP (in pel) */
  spHeight,     /* vertical size of SP (in pel) */
  lenTop,       /* byte length of the top    field compressed bitmap */
  lenBottom,    /* byte length of the bottom field compressed bitmap */
  startx;       /* start column of the SP on screen */
 Uint32
  offset;       /* byte offset relative to the start of the SPU */

 starty &= -1;  /* starty MUST be even */

 /* generate run-length code for top field: */
 lenTop   = makeCompressedBitmap( spUBuffer+4
                                  0,
                                  text,
```

【図16】

```
                                        8, 8, 3, 3,
                                        &spWide, &spHeight );

/* generate run-length code for bottom field: */
lenBottom = makeCompressedBitmap ( spuBuffer+4+lenTop,
                                        1,
                                        text,
                                        8, 8, 3, 3,
                                        &spWide, &spHeight );

/* place SP at horizontal screen center position: */
startx = (720 - spWide) / 2;

/* set offset to start of SP_DCSQT: */
offset = 4 + lenTop + lenBottom;

/* size of SP_DCSQT <= half of the size of SPU */
while (offset < 30 ) spuBuffer [ offset++ ] = 0;

spuBuffer[0] = ((offset+31)&-1) >> 8;      /* size of whole SPU */
spuBuffer[1] = ((offset+31)&-1) & 0xFF;

spuBuffer[2] = offset >> 8;                /* start of SP_DSQ #0 */
spuBuffer[3] = offset & 0xFF;

/*** DCSQ#0: ***/

spuBuffer[offset ] = 0x00;                 /* STM = 0 */
spuBuffer[offset+1] = 0x00;
spuBuffer[offset+2] = (offset+24) >> 8;    /* ptr to next SP_DCSQ */
spuBuffer[offset+3] = (offset+24) & 0xFF;

spuBuffer[offset+4] = 0x03;   /* SET_COLOR b=0 p=1 e1=2 e2=3 */
spuBuffer[offset+5] = 0x32;
spuBuffer[offset+6] = 0x10;

spuBuffer[offset+7] = 0x04;   /* SET_CONTR (0..15) b=15 p=15 e1=15 e2=15 */
spuBuffer[offset+8] = 0xFF;
spuBuffer[offset+9] = 0xFF;

spuBuffer[offset+10] = 0x05; /* SET_DAREA */
spuBuffer[offset+11] = startx>>4;
spuBuffer[offset+12] = ((startx<<4) +( (startx+spWide-1)>>8)) & 0xF3;
spuBuffer[offset+13] = (startx+spWide-1) & 0xFF;
spuBuffer[offset+14] = starty>>4;
spuBuffer[offset+15] = ((starty<<4)+((starty+spHeight-1)>>8)) & 0xE3;
spuBuffer[offset+16] = (starty+spHeight-1) & 0xFF;

spuBuffer[offset+17] = 0x06; /* SETDSPXA */
spuBuffer[offset+18] = 0x00;
spuBuffer[offset+19] = 0x04;
spuBuffer[offset+20] = (4+lenTop) >> 8;
spuBuffer[offset+21] = (4+lenTop) & 0xFF;

spuBuffer[offset+22] = 0x01; /* STA_DSP */

spuBuffer[offset+23] = 0xFF; /* CMD_END */

/*** SP_DCSQ#1: ***/

if( ntsc ) { /* STM for NTSC: */
```

【図17】

```
  spuBuffer[offset+24]  =  (durationInFrames * 3003) >> 18;
  spuBuffer[offset+25]  =  ((durationInFrames * 3003) >> 10) & 0xFF;
 } else {      /* STM for PAL: */
  spuBuffer[offset+24]  =  (durationInFrames * 225) >> 14;
  spuBuffer[offset+25]  =  ((durationInFrames * 225) >>  6) & 0xFF;

 spuBuffer[offset+26]  = spuBuffer[offset+2];   /* ptr to Sp_DCSQ#1 */
 spuBuffer[offset+27]  = spuBuffer[offset+3];

 spuBuffer[offset+28]  = 0x02;  /* STP_DSP   */

 spuBuffer[offset+29]  = 0xFF;  /* CMD_END */

 if( offset&1 )  spuBuffer[offset+30] = 0xFF;  /* stuffing if necessary */

 return (offset+31)  & ~1;     /* length of the whole SPU */

} /*   makeSpu( ) */
/*****************************************************************
 *
 *   Extern Function: makeSPPack( )
 *
 *   Generates a complete sub-picture pack.
 *   Input:  pointer to an already allocated buffer for the complete
 *           2048 main data
 *   Output:  the encoded SP pack
 *
 *   No return value
 *
 *   Note:  this simple version doesn't perform scrambled encoding
 *
 *****************************************************************/

extern void makeSPPack(
      Uint8  ntsc,              /* 0: 625/50;  1: 525/60 */
      Uint8  *packBuffer,       /* OUT: already allocated buffer for the SPU */
      char   *text,             /* this text shall be encoded as SP */
      uint32 durationInFrames,  /* after this no. of frames
                                 * the SP shall disappear */
      Uint32  starty,           /* vertical start position of the SP */
      Uint8  *scr,              /* 6 bytes SCR */
      Uint8  *pts,              /* 5 bytes PTS,must be on top field boundary!  */
      Uint8   firstPacket,      /* 1:first packet of a VOB,  else 0 */
      Uint8   spStreamNumber    /* SP stream number: 0..31 */
      ) {
 Uint32 spuSize;
 Uint32 pesPrivateDataSize=(firstPacket?3:0);

 /* insert pack header:  */
 packBuffer[0]  = 0x00;
 packBuffer[1]  = 0x00;
 packBuffer[2]  = 0x01;
 packBuffer[3]  = 0xBA;

 packBuffer[4]  = scr[0];
 packBuffer[5]  = scr[1];
 packBuffer[6]  = scr[2];
 packBuffer[7]  = scr[3];
 packBuffer[8]  = scr[4];
 packBuffer[9]  = scr[5];
```

【図18】

```
packBuffer[10]   = 0x01;
packBuffer[11]   = 0x89;
packBuffer[12]   = 0xC3;

packBuffer[13]   = 0xF8;
/*  insert SPU:     */
spuSize = makeSpU( ntsc,
                   packBuffer+14+14+1+pesPrivateDataSize,
                   text,
                   durationInFrames,
                   startY );

/* insert packet header:   */
packBuffer[14]   = 0x00;
packBuffer[15]   = 0x00;
packBuffer[16]   = 0x01;
packBuffer[17]   = 0xBD;

packBuffer[18]   = (spuSize+8+1+pesPrivateDataSize)  >>   8;
packBuffer[19]   = (spuSize+8+1+pesPrivateDataSize)  & 0xFF;

packBuffer[20]   = 0x81;
packBuffer[21]   = 0x80 + (firstpacket != 0);
packBuffer[22]   = 5+pesPrivateDataSize;

packBuffer[23]   = pts[0];
packBuffer[24]   = pts[1];
packBuffer[25]   = pts[2];
packBuffer[26]   = pts[3];
packBuffer[27]   = pts[4];

if( pesPrivateDataSize )  {
  packBuffer[28]   = 0x1E;
  packBuffer[29]   = 0x60;
  packBuffer[30]   = 58;


/* private data sub stream id;   */
packBuffer [28+pesPrivateDataSize]  = 0x20 + (spStreamNumber&0x1F);

/* clear unused sector rest:   */
while( spuSize + 14 + 14 + 1 + pesPrivateDataSize < 2048 )
  packBuffer[ spuSize++ + 14 + 14 + 1 + pesPrivateDataSize ]   = 0;
} /* makeSppack( )  */
```

D

【図19】

```
/*-----------------------------------------------------------------------------
| Copyright (c) 1998, This software is the property of Thomson
| multimedia, and shall not be reproduced, copied, or distributed
| without written permission
-----------------------------------------------------------------------------*/
/* :NTSC,
 * SP shall disappear after 123 frames,
 * vertical start position of the SP: 400,
 * scr contains the 6 bytes of SCR
 * pts contains the 5 bytes of PTS
 * it's the first SPU in this VOB,
 * it's SP stream #3 */
makeSPPack ( 1,     /* 0:   625/50;  1: 525/60  */
   mainData,                /* ptr to sector main data */
   "01 : 23 : 45 : 67 : 89",  /* this text shall be encoded as SP */
   123,                     /* after this no. of frames
                             * the SP shall disappear */
   400,                     /* vertical start position of the SP */
   scr,                     /* 6 bytes SCR */
   pts,                     /* 5 bytes PTS,must be on a top field! */
   1,                       /* 1:first packet of a VOB, else 0 */
   3) ;                     /* SP stream number: 0..31 */
/* PAL,
 * SP shall disappear after 50 frames,
 * vertical start position of the SP: 500,
 * scr contains the 6 bytes of SCR
 * pts contains the 5 bytes of PTS
 * it's NOT the first SPU in this VOB,
 * it's SP stream #31 */
makeSPPack ( 0,    /* 0: 625/50;    1:  525/60 */
   mainData,                /* ptr to sector main data */
   "0 1 2 3 4 5",           /* this text shall be encoded as SP */
   50,                      /* after this no. of frames
                             * the SP shall disappear */
   500,                     /* vertical start position of the SP*/
   scr,                     /* 6 bytes SCR */
   pts,                     /* 5 bytes PTS,must be on a top field! */
   0,                       /* 1:first packet of a VOB, else 0 */
   31);                     /* SP stream number: 0..31 */

/* NTSC,
 * SP shall disappear after 60 frames,
 * vertical Start Position of the SP:       420,
 * scr contains the 6 bytes of SCR
 * pts contains the 5 bytes of PTS
 * it's NOT the first SPU in this VOB,
 * it's SP stream #0 */
makeSPPack ( 1,     /* 0:   625/50; 1:  525/60 */
   mainData,                /* ptr to sector main data */
   "00 : 01 : 45",          /* this text shall be encoded as SP */
   60,                      /* after this no. of frames
                             * the SP shall disappear */
   420,                     /* vertical start position of the SP */
   scr,                     /* 6 bytes SCR */
   pts,                     /* 5 bytes PTS,must be on a top field! */
   0,                       /* 1:first packet of a VOB, else 0 */
   0);                      /* SP stream number: 0..31 */
```

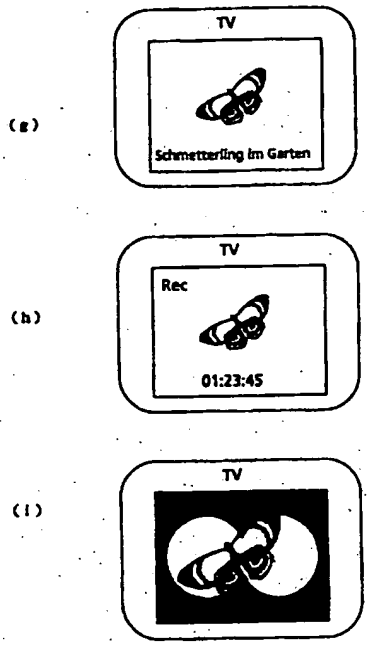
【図２０】

```
/* ------------------------------------------------------------------------
| Copyright (c) 1998, This text is the property of Thomson
| multimedia and shall not be reproduced, copied, or distributed
| without written permission.
------------------------------------------------------------------------ */

Note: values like 0xAB indicates hexadecimal values (C syntax)

/************************ THE FUNCTION CALL: ********************************/

/* NTSC
 * text:  "01:23",
 * SP shall be active for 123 frames
 * vertical start position of the SP:  400,
 * scr =  { 0x44, 0x00, 0x45, 0x46, 0xD4, 0xAB }
 * pts =  { 0x21, 0x00, 0x11, 0xDD, 0x09 }
 * it's the first SPU in this VOB.
 * it's SP stream #3 */
makeSPPack(
  1,                    /* 0: 625/50;  1: 525/60 */
  mainData,             /* ptr to sector main data */
  "01:23",              /* this text shall be encoded as SP */
  123,                  /* after this no.  of frames
                         * the SP shall disappear */
  400,                  /* vertical start position of the SP */
  scr,                  /* 6 bytes SCR */
  pts,                  /* 5 bytes PTS, must be on a top field!  */
  1,                    /* 1:first packet of a VOB,  else 0  */
  3);                   /* SP stream number:  0..31  */

/************************ THE RESULTING PACK:  ******************************/

hex-
addr  :     content of the pack (hex values)
------------------------------------------------------------------------------
0000000: 00 00 01 BA  44 00 45  46  D4  AB  01  89  C3  F8  00  00
0000010: 01 BD 00 76  81 81 08  21  00  11  DD  09  1E  60  3A  23
0000020: 00 6A 00 4C  00 00 00  00  20  58  5C  92  45  85  85  85
0000030: 20 20 58 51  05 2C 51  45  24  20  58  51  05  24  51  45
0000040: 85 20 00 00  00 00 00  00  24  91  45  28  91  09  24  20
0000050: 58 58 54 51  05 1C 51  45  20  20  58  51  05  10  51  45
0000060: 1C 52 00 24  91 0D 20  11  C9  24  00  00  00  00  00  64
0000070: 03 32 10 04  FF FF 05  15  21  7D  19  01  9C  06  00  04
0000080: 00 26 01 FF  01 68 00  64  02  FF  00  00  00  00  00  00
0000090: 00 00 00 00  00 00 00  00  00  00  00  00  00  00  00  00
00000A0: 00 00 00 00  00 00 00  00  00  00  00  00  00  00  00  00
00000B0: 00 00 00 00  00 00 00  00  00  00  00  00  00  00  00  00

... (the remaining bytes ... byte #2048 of the pack are all Zero)
```

【図２１】

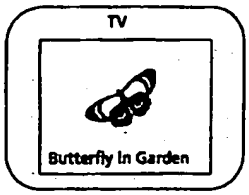
```
/*****************  ONLY THE SPU IN THE PACK:  *************************/

hex
addr:  content of the pack  (hex values)
---------------------------------

0020: ######################## SPUH:
0020: 00 6A                    #  SPDSZ     = 106 = 0x6A
0022: 00 4C                    #  SP_DCSQTA = 76 = 0x4C

0024: ######################## PXD:
0024: 00 00 00 00  20  58 5C 92 45  85  85  85 20  20  58  51
0034: 05 2C 51 45  24  20 58 51 05  24  51  45 85  20  00  00
0044: 00 00 00 00  24  91 45 28 91  09  24  20 58  58  54  51
0054: 05 1C  51 45  20  20 58 51 05  10  51  45 1C  52  00  24
0064: 91 0D 20 11  C9  24 00 00

006c: ######################## SP_DCSQT:
006c: ######################## SP_DCSQ 0 = 0x0
006C:  00 00                   # SP_DCSQ_STM = 0       = 0x0 = 0 0000s  => start frame:  0
006E:  00 64                   # SP_NXT_DCSQ_SA = 100 = 0x64
0070:  03  32  10              # SET_COLOR bg=0 p=1 e1=2 e2=3
0073:  04  FF  FF              # SET_CONTR  (0..15) b=15 p=15 e1=15 e2=15
0076: 05 15 21 7D 19 01 9C     # SET_DAREA sx=338 ex=381 sy=400 ey=412
007D:  06 00 04 00 26          # SET_DSPXA uf=0x4     bf=0x26
0082:  01                      # STA_DSP
0083:  FF                      # CMD_END
0084: ######################## SP_DCSQ 1 = 0x1
0084:  01  80                  # SP_DCSQ_STM = 0x180 = 122.8800s
0084:                          #       => start frame:  123  ( PAL )
0086:  00 64                   # SP_NXT_DCSQ_SA = 100 = 0x64
0088:  02                      # STP_DSP
0089:  FF                      # CMD_END

/*******************  THE RESULTING DECOMPRESSED SP:  ********************/

The following simple graphic shows the presented (decoded) bitmap.  The symbol
'.' indicates a background pixel and the symbol '#' indicates a pattern
pixel.  The following bitmap represents a sub-picture with horizontal 44 pixel
and vertical 13 pixel.

The (simple)  graphic:

.....  .....  .....
.....  .....  .....
.....  .....  .....
.....##.....#.....##.....##.....  .....
.....#...#...##.....#...#...#...#.....
.....#...#.#...#.....#.....#.....
.....#...#.....#.....#.....#.....
.....#...#.....#.....#.....#.....
.....#...#.....#.....#.....#.....
.....##.....###.....####....##.....
.....  .....  .....
.....  .....  .....
.....  .....  .....
.....  .....  .....
.....  .....  .....
.....  .....  .....
.....  .....  .....
```

【図２４】

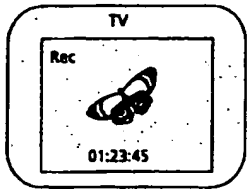


【手続補正書】
【提出日】平成１１年6月2日（１９９９．６．２）
【手続補正１】
【補正対象書類名】図面
【補正対象項目名】図２４
【補正方法】変更
【補正内容】
【図２４】

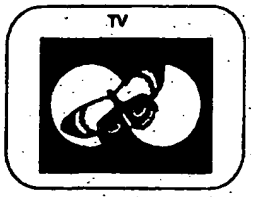
(a)



(b)



(c)



フロントページの続き

(72)発明者　マルコ　ヴィンター
ドイツ連邦共和国，30173　ハノーヴァー，
ベーマーシュトラーセ　17